Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 1er. Semestre 2023.

PROYECTO 1

202112395 – Evelyn Marisol Pumay Soy

Resumen

Durante la realización del programa se utilizó el lenguaje de programación Python el cual se encarga de leer datos de muestras de organismos vivos almacenados en un archivo XML, los procesa siguiendo una regla de prosperidad y los grafica a través de Graphviz. Se utiliza el paradigma de programación orientado a objetos (POO), definiendo clases para describir la estructura y el comportamiento de los objetos y crear instancias de ellas para representar objetos individuales en el programa los cuales corresponden a los organismos vivos dentro de cada muestras haciendo uso TDAs para la creación de una matriz dispersa la cual almacenaria cada uno de los datos. Se utilizaron estructuras de programación secuenciales, cíclicas y condicionales para el manejo de los datos, permitiendo la repetición de bloques de código, la ejecución de un bloque de código si se cumple una condición y el control de distintos casos. La herramienta Graphviz se utiliza para visualizar de forma gráfica los datos almacenados y leídos en el archivo XML.

Palabras clave

programación orientada a objetos, TDAs, paradigmas de programación, Graphviz, Listas enlazadas.

Abstract

During the program's implementation, the Python programming language was used to read data from samples of living organisms stored in an XML file, process them according to a prosperity rule, and graph them through Graphviz. The object-oriented programming (OOP) paradigm was used, defining classes to describe the structure and behavior of objects and creating instances of them to represent individual objects in the program, which correspond to the living organisms within each sample using ADTs for the creation of a sparse matrix that would store each of the data. Sequential, cyclic, and conditional programming structures were used to handle the data, allowing for the repetition of code blocks, execution of a code block if a condition is met, and control of different cases. The Graphviz tool is used to visually display the data stored and read in the XML file.

Keywords

Object-oriented programming, ADTs, programming paradigms, Graphviz, Linked lists.

Introducción

Para la resolución del programa planteado se utilizó el lenguaje de programación Python el cual es una de las opciones más populares y versátiles en la actualidad. Durante la realización del programa se utilizó el paradigma de programación orientado a objetos y también se aplicaron conceptos avanzados de memoria dinámica a través de la implementación de TDAs, en específico en la utilización de listas enlazadas para la creación de una lista dispersa. Además, se implementó la herramienta Graphviz para visualizar de forma gráfica los datos almacenados y leídos en un archivo XML, el cual permitió una entrada fácil, rápida y organizada de los datos necesarios en el programa.

Desarrollo del tema

El programa desarrollado solicitaba la lectura de un archivo XML, este archivo contenía datos de muestras y cada muestra contenía datos de organismos vivos, los cuales tenia que ser insertados siguiendo una regla de prosperidad, los datos debían ser graficados a través de Graphviz y al finalizar el programa se debía devolver el archivo XML con los cambios realizados.

a. Paradigma de programación orientado a objetos

El paradigma de programación orientada a objetos (POO) se basa en la idea de que los programas informáticos pueden modelar el mundo real mediante la creación de objetos que interactúan entre sí. En POO, se definen clases que describen la estructura y el comportamiento de los objetos, y se crean instancias de estas clases para representar objetos individuales en el programa.

En una matriz dispersa con listas enlazadas, POO se puede utilizar para representar cada elemento de la matriz como un objeto. Cada objeto tendría una posición en la matriz y un valor asociado. Además, cada objeto tendría punteros a los objetos vecinos (arriba, abajo, izquierda y derecha), lo que permitiría el acceso rápido a los elementos adyacentes.

La ventaja de utilizar POO en una matriz dispersa con listas enlazadas es que permite encapsular la lógica de la matriz en objetos individuales, lo que facilita la creación y manipulación de la matriz. Además, la POO permite la reutilización de código, lo que significa que el mismo código se puede utilizar para trabajar con matrices dispersas de diferentes tamaños y formas.

b. Uso de TDAs: matriz dispersa y listas enlazadas

Como se menciono anteriormente para almacenar los valores de cada uno de los organismos vivos en la muestra se utilizaron 2 TDAs: una matriz dispersa y listas enlazadas para la creación de la matriz. Cada elemento de la matriz se representó como un objeto que contiene su valor, su posición en la matriz y punteros a los elementos adyacentes, utilizando un TDA de matriz dispersa. Luego, estos objetos se almacenaron en una lista enlazada, utilizando un TDA de lista enlazada, para permitir un acceso rápido a los elementos y la creación eficiente de la matriz.

Se creó una clase "NodoInterno" para representar cada elemento de la matriz, que tendría atributos como el valor del elemento, su posición en la matriz y punteros a los elementos adyacentes. Luego, estos objetos "Nodo" se podrían almacenar en una lista enlazada, en la que cada nodo tendría un puntero al siguiente nodo en la lista.

Para acceder a un elemento en la matriz dispersa, se podría buscar su posición en la lista enlazada y devolver su valor. Si el elemento no está en la lista, su valor sería cero o nulo.

c. Estructuras de programación secuenciales, cíclicas y condicionales.

Las estructuras de programación secuenciales son aquellas que se ejecutan en orden secuencial, es decir, una tras otra. Un ejemplo de estructura de programación secuencial en Python es la asignación de variables o la ejecución de una función.

Por ejemplo, en la opción 1 del menú principal, se carga un archivo, se lee el archivo y se almacenan los datos en variables.

Las estructuras de programación cíclicas permiten repetir un bloque de código varias veces. En Python, las estructuras de programación cíclicas se dividen en dos tipos: while y for. En la estructura while, se repite un bloque de código mientras se cumpla una condición. Por otro lado, en la estructura for, se repite un bloque de código por cada elemento de una lista o de un rango de valores.

Un ejemplo de su uso en la programación seria en la opción 2 del menú principal, donde se muestra un listado de muestras utilizando un for, se pide al usuario que seleccione una y se trabajará con ella hasta que se seleccione la opción de salida.

Por último, las estructuras de programación condicionales permiten ejecutar un bloque de código si se cumple una condición. En Python, la estructura de programación condicional más común es el if. Esta estructura permite ejecutar un bloque de código si se cumple una condición determinada. Además, se pueden añadir estructuras de programación else y elif para controlar distintos casos de condiciones.

Por ejemplo, en la opción 2 del menú, se presenta un submenú con diferentes opciones y se ejecuta la opción seleccionada por el usuario.

d. Lectura de archivo XML

Un archivo XML (Extensible Markup Language) es un formato de archivo utilizado para almacenar datos estructurados y jerarquizados. Es un lenguaje de marcado que define etiquetas personalizadas para describir datos y sus relaciones.

En un archivo XML, los datos se organizan en forma de árbol, donde cada elemento tiene una etiqueta y un valor asociado. Los elementos pueden tener atributos que proporcionan información adicional sobre el elemento, como su tipo o formato.

Durante la realización del programa se utilizó la biblioteca "ElementTree" de Python para analizar las cadenas de textos del archivo XML y convertirlas en estructuras de datos y así acceder a los elementos y atributos del archivo. Los datos se extrajeron del archivo XML y se utilizaron para crear objetos de las clases "muestra" y "celdas_vivas".

e. Manejo de Graphviz

Graphviz es una herramienta de visualización de gráficos y diagramas que permite crear gráficos en una variedad de formatos, incluyendo PNG, PDF y SVG, entre otros. Se utiliza comúnmente para crear diagramas de flujo, gráficos de red, árboles genealógicos, diagramas de entidad-relación y otros tipos de diagramas.

Graphviz utiliza un lenguaje de descripción de gráficos llamado DOT, que permite especificar los nodos, aristas y otras características de los gráficos de manera estructurada. Luego, la herramienta de Graphviz interpreta este código y produce un gráfico que representa visualmente la estructura de los datos.

Conclusiones

La implementación de TDAs en conjunto con el paradigma de programación orientada a objetos tiene varias ventajas, como la encapsulación de la lógica en objetos individuales y la reutilización de código. En este proyecto, se utilizó una matriz dispersa con listas enlazadas para almacenar los valores de cada organismo vivo en una muestra. Cada elemento de la matriz se representó como un objeto con un valor, una posición en la matriz y punteros a los elementos adyacentes.

La herramienta Graphviz también fue utilizada para visualizar los datos almacenados y leídos en un archivo XML, lo que permitió una visualización fácil, rápida y organizada de los datos en el programa.

En cuanto a las estructuras de programación, se utilizaron estructuras secuenciales, cíclicas y condicionales. Las estructuras secuenciales se utilizaron para asignar variables y ejecutar funciones en orden secuencial. Las estructuras cíclicas se utilizaron para repetir bloques de código varias veces, mientras que las estructuras condicionales permitieron ejecutar bloques de código si se cumplía una condición.

Anexos

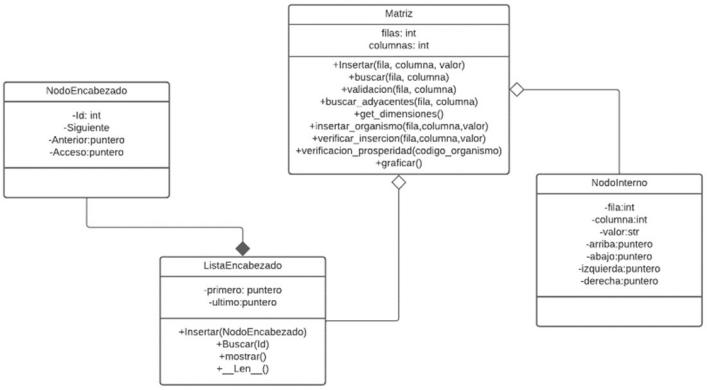

Figura 1. Diagrama de clases matriz dispersa

Fuente: elaboración propia, 2023.

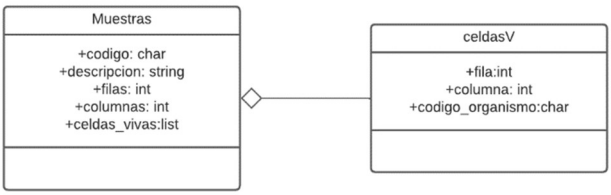

Figura 2. Diagrama de clases matriz dispersa

Fuente: elaboración propia, 2023.

```
class muestra:
    def __init__(self, codigo, descripcion, filas, columnas, celdas_vivas):
        self.codigo = codigo
        self.descripcion = descripcion
        self.filas = filas
        self.columnas=columnas
        self.celdas_vivas=celdas_vivas

class celdasV:
    def __init__(self, fila, columna, codigo_organismo):
        self.fila = fila
        self.columna=columna
        self.codigo_organismo = codigo_organismo
```

Figura 3. Definición de clase muestras y clase celdasV.

Fuente: elaboración propia, 2023.

```python
def graficar(self):
    if self.filas.primero == None:
        return
    if self.columnas.primero == None:
        return

    file = open("Matriz.dot", "w")
    file.write("digraph G{\n")
    file.write("node [shape=plaintext];\n")
    file.write("rankdir=LR;\n")
    file.write("Matriz [\n")
    file.write("label=<<table border='0' cellborder='1' cellspacing='0'> \n")

    file.write("<tr>\n")
    file.write("<td></td>\n")

    actual = self.columnas.primero
    while actual != None:
        file.write("<td bgcolor=\"gray87\">" + str(actual.id) + "</td>\n")
        actual = actual.siguiente
    file.write("</tr>\n")

    actual = self.filas.primero
    while actual != None:
        file.write("<tr>\n")
        file.write("<td bgcolor=\"gray87\">" + str(actual.id) + "</td>\n")

        aux = actual.acceso
        actual_col = self.columnas.primero
        while actual_col != None:
            if aux != None and aux.col == actual_col.id:
                if aux.valor == "":
```

Figura 4. método graficar utilizando Graphviz.

Fuente: elaboración propia, 2023.

```python
def insertar_organismo(self, fila, columna,valor_encerrar):
    celdas_adyacentes=self.buscar_adyacentes(fila, columna)
    if celdas_adyacentes==[]:
        print("El organismo no prospera")

    celdas_diferentes=[]
    for celda in celdas_adyacentes:
        if celda.codigo_organismo!=valor_encerrar:
            celdas_diferentes.append(celda)
    filas_matriz, columnas_matriz=self.get_dimensiones()

    for organismo in celdas_diferentes:
        encerradas=[]

        valor_encerrado=organismo.codigo_organismo
        direccion_fila=organismo.fila-fila
        direccion_col=organismo.columna-columna
        i = fila
        j = columna
        while i >= 0 and i < filas_matriz and j >= 0 and j <columnas_matriz :
            i += direccion_fila
            j += direccion_col
            valor_celda=self.buscar(i, j).valor
            if valor_celda==None:
                print("El organismo no prospera")
                break
            elif valor_celda==valor_encerrado:
                celdas_encerradas=celdasV(i, j, valor_celda)
                encerradas.append(celdas_encerradas)
            elif valor_celda==valor_encerrar:
                for encerrada in encerradas:
                    insertada=self.buscar(fila,columna)
```

Figura 5. Método de inserción de organismo.

Fuente: elaboración propia, 2023.

Referencias bibliográficas

Lutz, M. (2013). Learning Python, 5th Edition. O'Reilly Media.

McKinney, W. (2017). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly Media.